Diffusion model based image generation and model maintenance

Aryansh Saxena, Bijayan Ray, Krishanu Bandyopadhyay

May 8, 2025

Contents

Dataset preparation I

- A custom dataset class named `FlowerDataset` is defined, inheriting from `torch.utils.data.Dataset`.
- The constructor takes the directory path of the images and an optional transform.
- It lists all files in the given directory that have a `.jpg` extension.
- Class names are extracted from each file name by splitting at the underscore (assuming file names are formatted as `class_index.jpg`).
- A sorted list of unique class names is created, and a mapping from class names to numeric indices is generated.
- The `__len__` method returns the number of image files in the dataset.

Dataset preparation II

- The `__getitem__` method opens an image file, converts it to RGB format, applies the transform (if provided), extracts the class label from the file name, maps it to its corresponding index, and returns the image and its label.
- An instance of the dataset is created using the image directory `/kaggle/input/flower-image-dataset/flowers` and a specified transform.
- A `DataLoader` is created with the dataset, a specified batch size, shuffling enabled, and 2 worker processes for data loading.

Diffusion model description (simple unet version) I

- The model is a U-Net-style architecture with both time and text conditioning, suitable for denoising in diffusion models. The input is a noisy image $x \in \mathbb{R}^{B \times C \times H \times W}$, a timestep $t \in \mathbb{N}^B$, and an optional text embedding $e_{\text{text}} \in \mathbb{R}^{B \times d_{\text{text}}}$.
- Time embedding block:
  - A learnable positional embedding layer maps timestep t to an embedding vector:

$$\text{Emb}(t) \in \mathbb{R}^{B \times d_{\text{time}}}$$

    where $\text{Emb} : \mathbb{N} \to \mathbb{R}^{d_{\text{time}}}$ is a learned embedding function.
  - The time embedding is passed through a multilayer perceptron (MLP) consisting of:
    - A linear layer: $W_1 \in \mathbb{R}^{4d_{\text{time}} \times d_{\text{time}}}$
    - SiLU activation: $\text{SiLU}(x) = x \cdot \sigma(x)$
    - Another linear layer: $W_2 \in \mathbb{R}^{d_{\text{time}} \times 4d_{\text{time}}}$
  - Output:

$$e_t = W_2 \cdot \text{SiLU}(W_1 \cdot \text{Emb}(t)) \in \mathbb{R}^{B \times d_{\text{time}}}$$

Diffusion model description (simple unet version) II

- Text conditioning:
  - If available, text embedding $e_{\text{text}} \in \mathbb{R}^{B \times d_{\text{text}}}$ is linearly projected:
$$e'_{\text{text}} = W_{\text{text}} \cdot e_{\text{text}} \in \mathbb{R}^{B \times d_{\text{time}}}$$
  - Combined with the time embedding as:
$$e_{\text{cond}} = e_t + e'_{\text{text}}$$
- Residual block structure:
  - Input: $x \in \mathbb{R}^{B \times C_{\text{in}} \times H \times W}$ and $e_{\text{cond}} \in \mathbb{R}^{B \times d_{\text{time}}}$
  - Time embedding projection:
$$e_{\text{proj}} = W_t \cdot e_{\text{cond}} \in \mathbb{R}^{B \times C_{\text{out}}}$$
  - Two convolutional layers with GroupNorm and SiLU:
    - First layer: GroupNorm $\rightarrow$ SiLU $\rightarrow$ Conv2d($C_{\text{in}} \rightarrow C_{\text{out}}$)
    - Second layer: GroupNorm $\rightarrow$ SiLU $\rightarrow$ Conv2d($C_{\text{out}} \rightarrow C_{\text{out}}$)

Diffusion model description (simple unet version) III

- After the second convolution, add time embedding:

$$h = h + e_{\text{proj}}[:, :, None, None]$$

- Residual connection:
  - If $C_{\text{in}} \neq C_{\text{out}}$: apply 1×1 convolution.
  - Otherwise, use identity.
- Final output:

$$y = h + \text{ResConv}(x)$$

- Encoder pathway (downsampling):
  - Input x passes through the first residual block:

$$d_1 = \text{ResBlock}_1(x, e_{\text{cond}})$$

  - Max pooling by a 2×2 kernel:

$$d_1' = \text{MaxPool}(d_1)$$

Diffusion model description (simple unet version) IV

- Pass through second residual block:

$$d_2 = \text{ResBlock}_2(d_1', e_{\text{cond}})$$

- Another downsampling step:

$$d_2' = \text{MaxPool}(d_2)$$

- Bottleneck:
  - Apply one residual block at the lowest resolution:

$$b = \text{ResBlock}_{\text{bot}}(d_2', e_{\text{cond}})$$

- Decoder pathway (upsampling):
  - Upsample the bottleneck output:

$$u_1' = \text{Upsample}(b)$$

  - Concatenate with d_2 along channel dimension:

$$u_1'' = \text{Concat}(u_1', d_2) \in \mathbb{R}^{B \times (C_b + C_{d_2}) \times H \times W}$$

Diffusion model description (simple unet version) V

- Apply residual block:

$$u_1 = \text{ResBlock}_3(u_1'', e_{\text{cond}})$$

- Repeat upsampling:

$$u_2' = \text{Upsample}(u_1)$$

- Concatenate with d_1 and apply final residual block:

$$u_2 = \text{ResBlock}_4(\text{Concat}(u_2', d_1), e_{\text{cond}})$$

- Output head:
  - GroupNorm $\rightarrow$ SiLU $\rightarrow$ Conv2d with kernel size 1×1:

$$\hat{x} = \text{Conv2d}_{1\times1}(\text{SiLU}(\text{GroupNorm}(u_2))) \in \mathbb{R}^{B\times C\times H\times W}$$

Diffusion model description (enhanced model version) I

- Sinusoidal Positional Embedding:
  - The time embedding is generated using sine and cosine functions.
  - Frequencies are defined as:

$$f_i = 10000^{-\frac{2i}{d}}$$

    where i is the dimension index, and d is the total dimensionality.
  - The time embedding for timestep t is computed as:

$$\text{emb}(t) = \left[\sin(t \cdot f_1), \cos(t \cdot f_1), \ldots, \sin(t \cdot f_{d/2}), \cos(t \cdot f_{d/2})\right]$$

  - This embedding encodes timestep information, enabling the network to understand temporal dependencies during training.
- Conditional Dropout:
  - Dropout is applied during training to regularize the model.
  - A mask is created based on the dropout probability p, and features are zeroed out accordingly.

Diffusion model description (enhanced model version) II

- Dropout is not applied during inference; the model behaves deterministically during testing.
- If the dropout probability p is zero, the input embeddings remain unchanged.
- This technique prevents overfitting by forcing the model to rely less on specific features during training.

- Residual Blocks:
  - A residual block consists of a series of operations designed to improve gradient flow.
  - It contains a group normalization layer followed by a convolutional layer.
  - After the convolution, a non-linearity (SiLU) is applied.
  - A second convolutional layer is applied to refine the features.
  - The input x is added back to the output of the convolutions (skip connection):

$$\text{output} = \text{ResidualBlock}(x) = \text{Conv}(\text{Norm}(x)) + x$$

Diffusion model description (enhanced model version) III

- Skip connections mitigate the vanishing gradient problem, improving model training.

- Cross Attention:
  - Cross-attention allows the model to condition on text information while processing image features.
  - The query Q, key K, and value V are derived from the image features and text embeddings.
  - The attention scores are computed by taking the dot product between the query and key, scaled by $\sqrt{d}$, where d is the dimension of the key:

$$\text{attn\_scores} = \frac{QK^T}{\sqrt{d}}$$

  - Softmax is applied to the attention scores to normalize them.
  - The attention output is computed as the weighted sum of the value matrix V:

$$\text{attn\_output} = \text{Softmax}(\text{attn\_scores})V$$

Diffusion model description (enhanced model version) IV

- This output is used to refine the image features by incorporating text-driven information.

- Self Attention:
  - Self-attention allows each pixel in the image to attend to every other pixel, capturing long-range dependencies.
  - The query, key, and value matrices are derived from the image features.
  - Attention scores are computed in the same manner as cross-attention:

$$\text{attn\_scores} = \frac{QK^T}{\sqrt{d}}$$

  - Softmax is applied to the scores, and the output is computed as:

$$\text{attn\_output} = \text{Softmax}(\text{attn\_scores})V$$

  - The output is reshaped back into the original image dimensions and passed through a projection layer.
  - The projection layer ensures that the output matches the expected number of channels.

Diffusion model description (enhanced model version) V

- DownBlock:
  - The DownBlock is a part of the encoder, which reduces the spatial dimensions of the image.
  - It consists of a residual block followed by an optional cross-attention layer.
  - After the residual and attention layers, a second residual block is applied to refine the features.
  - The final operation in the DownBlock is a convolution with stride 2, which reduces the image dimensions by half.
  - The output consists of the processed feature map and the downsampled version of the feature map.
- UpBlock:
  - The UpBlock is part of the decoder and is responsible for increasing the spatial resolution of the feature map.
  - The first operation is a transposed convolution, which upsamples the feature map.

Diffusion model description (enhanced model version) VI

- The upsampled feature map is concatenated with the corresponding skip connection from the encoder to retain fine-grained spatial details.
- The concatenated feature map is passed through a residual block and an optional cross-attention layer.
- A second residual block is applied to further refine the features.
- The final output of the UpBlock is the refined feature map after upsampling.

- TextGuidedUNet:
  - The TextGuidedUNet extends the standard UNet by incorporating text-guided conditioning during image generation.
  - The model consists of an encoder-decoder architecture with additional modules for time embedding and text-based attention.
  - In the encoder, two DownBlocks are used to process the image and progressively downsample it.

Diffusion model description (enhanced model version) VII

- The bottleneck includes residual blocks and self-attention to capture global dependencies.
- The decoder uses two UpBlocks to restore the spatial resolution of the image.
- Skip connections are used to preserve high-resolution details from the encoder.
- The final output is generated through a 1×1 convolutional layer, reducing the feature map to the desired output channels.

- Forward Pass:
  - During the forward pass, the model first computes time embeddings for the input timestep t using the sinusoidal positional embedding.
  - The text embedding is conditioned using the conditional dropout mechanism, which randomly drops certain features during training.
  - The image is processed through the encoder (comprising DownBlocks) and the bottleneck, where residual and self-attention layers are applied.

Diffusion model description (enhanced model version) VIII

- The feature map is passed through the decoder (comprising UpBlocks), where cross-attention and residual processing are used.
- Skip connections from the encoder are concatenated with the upsampled feature maps during decoding to retain fine-grained spatial information.
- The final output is produced by a 1×1 convolution, which maps the features back to the image space (e.g., RGB channels).

Configurations I

- The computation device is set to use CUDA if available; otherwise, it falls back to CPU.
- The batch size for training is set to 128.
- The number of training epochs is 45.
- The learning rate is set to 2×10^{-4}.
- The input image size is set to 56 pixels.
- The number of image channels is 3, corresponding to RGB images.
- The number of diffusion timesteps T is set to 1000.
- The dimensionality of the time embedding is 128.
- The dimensionality of the text embedding is 512.
- The conditional dropout probability is set to 0.1.

Model training I

- Initialization:
  - Initialize empty lists:

$$\text{loss\_hist} \leftarrow [], \quad \text{acc\_hist} \leftarrow []$$

  - Loop over epochs:

$$\text{for epoch} \in \{0, 1, \ldots, \text{epochs} - 1\}$$

- Set model to training mode:

$$\text{model.train()}$$

  This enables gradient computation and training-specific layers like dropout or batch normalization (if present).
- Initialize epoch statistics:
  - Total loss accumulator: $\text{total\_loss} \leftarrow 0$
  - Total MSE accumulator for pseudo-accuracy: $\text{total\_mse} \leftarrow 0$

Model training II

- Iterate over training batches:

$$\text{for } (\text{imgs}, \text{labels}) \in \text{train\_loader}$$

  - Transfer input images to device:

$$\text{imgs} \leftarrow \text{imgs.to(device)}$$

  - Compute batch size:

$$b \leftarrow \text{imgs.size}(0)$$

  - Sample random diffusion timesteps:

$$t \sim \text{Uniform}\{0, T-1\}^b$$

  - Generate noise tensor:

$$\epsilon \sim \mathcal{N}(0, I), \quad \epsilon \in \mathbb{R}^{B \times C \times H \times W}$$

Model training III

- Compute $\bar{\alpha}_t$ from precomputed α_{cumprod}:

$$\bar{\alpha}_t = \text{extract}(\alpha_{\text{cumprod}}, t, \text{imgs.shape})$$

- Corrupt input image using forward diffusion process:

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$$

- Get text embeddings for labels:

$$e_{\text{text}} = \text{label\_embs}[\text{labels}]$$

- Predict noise:

$$\hat{\epsilon} = \text{model}(x_t, t, e_{\text{text}})$$

- Compute loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{BCHW} \sum_{i=1}^{B} \|\hat{\epsilon}_i - \epsilon_i\|^2$$

where B is batch size, C is channels, H is height and W is width.

Model training IV

- Backpropagation and optimization:
  - Zero gradients: opt.zero_grad()
  - Compute gradients: $\mathcal{L}_{\text{MSE}}$.backward()
  - Update weights: opt.step()
- Accumulate total loss:

$$\text{total\_loss} \mathrel{+}= \mathcal{L}_{\text{MSE}} \cdot b$$

- Estimate denoised image x_0 using reparameterization:

$$\hat{x}_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}$$

- Compute pseudo-accuracy =1 - MSE where MSE is:

$$\text{MSE} = \frac{1}{BCHW} \sum_{i=1}^{B} \|\hat{x}_0^{(i)} - x^{(i)}\|^2$$

Model training V

- Accumulate MSE:

$$\text{total\_mse} \mathrel{+}= \text{MSE} \cdot b$$

- End of epoch:
  - Normalize accumulated loss and accuracy:

$$\text{epoch\_loss} = \frac{\text{total\_loss}}{N}, \quad \text{epoch\_acc} = 1 - \frac{\text{total\_mse}}{N}$$

    where $N = \text{len}(\text{train\_ds})$
  - Append to history:

$$\text{loss\_hist.append}(\text{epoch\_loss}), \quad \text{acc\_hist.append}(\text{epoch\_acc})$$

  - Log accuracy to file:

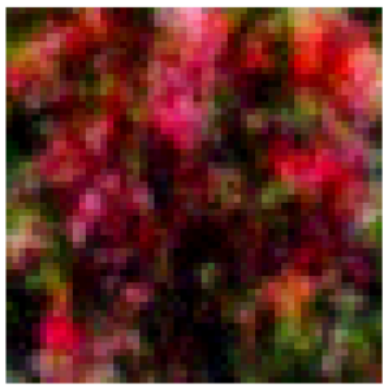
    Append "*Epoch#i* : *Accuracy* $\approx x$" to `accuracy_log.txt`

Results I

Some images generated from the diffusion model trained on flower dataset:

Orchids

Orchids



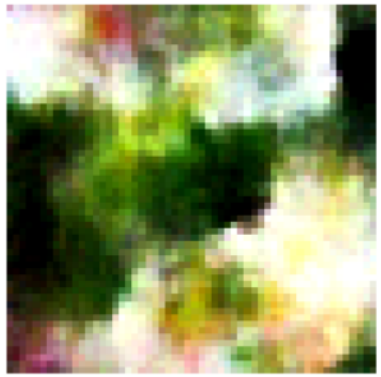
Results II

Orchids

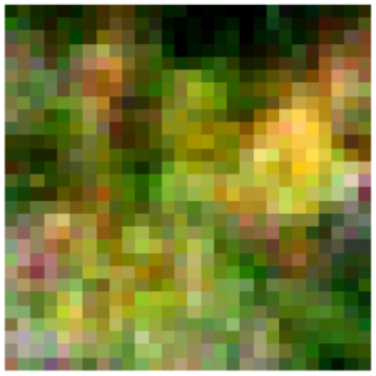
Orchids

Results III

Orchids

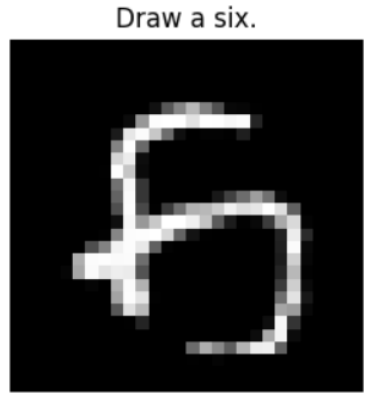


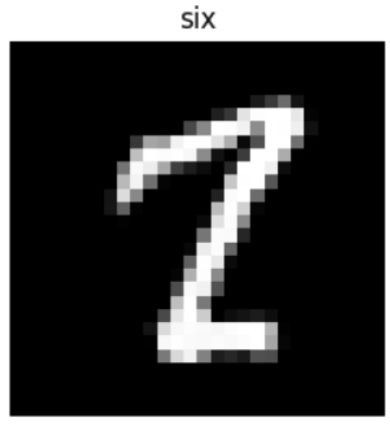
Draw a tulips



Some images generated from the diffusion model trained on MNIST dataset:

Results IV





Summary I

- This work demonstrates a complete implementation of a text-conditioned diffusion model, enabling the generation of class-specific images purely from natural language prompts.
- The architecture is built around a UNet backbone enhanced with both cross-attention for text guidance and self-attention for better spatial consistency.
- CLIP's pretrained text encoder ensures semantically rich and robust embeddings, making the model responsive to nuanced textual descriptions.
- Training leverages a simplified loss function using noise prediction, aligning with the DDPM framework, and incorporates classifier-free guidance during inference for improved sample quality.
- The modular design supports extensibility, such as replacing the text encoder or scaling the model to higher resolutions and more complex datasets.

Summary II

- Overall, this implementation serves as a foundational and interpretable framework for understanding and building text-to-image diffusion models in research or production settings.

Thank you!